AERONAUTICAL SYSTEMS CENTER
MAJOR SHARED RESOURCE CENTER

IBM P3
USER'S GUIDE

7 July 2003

Release Notes

Issued January, 2000.

Issued June, 2000.

Issued March 2003.
- Changed from IBM SP P3 to IBM SP.
- Updated software information.

Issued June 2003.
- Changed reference from IBM SP to IBM P3.

Issued July 2003
- Cleaned up formatting.
- Removed explicit queue class information.

Table of Contents

1. Introduction

This document provides an overview and introduction to using the IBM P3 system, located at the Aeronautical Systems Center (ASC) Major Shared Resource Center (MSRC). This guide is intended to provide information so that users who are familiar with the UNIX operating system can create and run their own programs, as well as use existing application software on the IBM P3 system.

1.1 Assumed Background of the Reader

It is assumed that the reader of this guide has a firm grasp of the concepts required to use the UNIX operating system and to program in the C, C++, Fortran 77, Fortran 90, or Fortran 95 languages. It is also assumed that the reader has read the *ASC MSRC User's Guide*, which contains site specific information about the ASC MSRC. The *ASC MSRC User's Guide* is available from the ASC MSRC Service Center at 1-888-MSRC-ASC (1-888-677-2272), (937) 255-0194 or DSN 785-0194. It is also available in PostScript and PDF formats as described in Section 1.7. UNIX training is periodically offered by the ASC MSRC.

1.2 Hardware Overview

The IBM P3 system is a scalable distributed memory parallel computer based on the IBM RS/6000 processor with 132 symmetric multi-processor (SMP) nodes. Currently each node has four processors. Each node on the IBM P3 has 4 Gigabytes (GB) of memory and 1 GB of local diskspace. An additional 2.4 Terabytes (TB) of diskspace is available on the `/workspace` filesystem which is accessible from all the nodes.

1.3 Accessing the IBM P3

The fully qualified hostnames of the interactive nodes are *hpc04-1.asc.hpc.mil* and *hpc04-2.asc.hpc.mil*.

Users are only allowed to login onto the interactive nodes of the system. The other nodes are for batch jobs only. Users submit their jobs on the front-end nodes and the batch system will automatically start their jobs on the other systems based on the load of the system. See the *ASC MSRC User's Guide* for instructions on accessing the machines.

1.4 ASC MSRC Connectivity

Since the IBM P3 is an integrated component of the ASC MSRC, user files are Network File System **(**NFS) mounted from an ASC MSRC High Availability File Server (HAFS) system to the IBM P3. When users log into a system,, their home ($HOME) directory (which will be the current directory immediately after logging in) physically resides on the file server, but appears to be on the IBM P3. The ASC MSRC also supplies archival storage and visualization capabilities.

1.5 ASC MSRC Startup Files

All users are provided a `.cshrc` and `.login` file in their home directory. These files reference a standard setup file, maintained by the site administrators in a central location, which set up a standard environment for all MSRC users. These file **should not** be modified.

To set up specific information for your IBM P3 session, such as environment variables, path information, terminal information, or command aliases, place the appropriate commands and information into files called `.personal.cshrc` and `.personal.login`. The standard startup files check your home directory for the existence of these files and executes them if found. Commands related to aliases, prompts, and some environment variables should go into `.personal.cshrc`, while commands related to the type of terminal you are using should go into `.personal.login`. See Section 3 of this guide for more details on the IBM P3 computing environment and the *ASC MSRC User's Guide* for more details on startup files.

1.6 The Archive Command

The **archive** command is a recently added tool to the ASC MSRC to help users with transferring files to and from $ARC. The basic syntax for the archive command is:

archive get [getopts] file1 [file2 ...]
archive put [putopts] file1 [file2 ...]

More information on the **archive** command can be found online via the archive man page (***man archive***).

1.7 Additional Information

Much of the information presented in this document is available on-line through man pages and is accessible by typing:

man {command name}

when logged into the IBM P3.

The *ASC MSRC User's Guide* and this document are all available in PDF and PostScript format. They may be downloaded from the ASC MSRC website (http://www.asc.hpc.mil/customer/userdocs).

2. ASC MSRC IBM P3

This section details the hardware and software available on the IBM P3 and how they are currently configured.

2.1 Hardware

The IBM P3 is a parallel computer supporting SPMD (single program, multiple data) and MPMD (multiple program, multiple data) models. The IBM P3 has 132 symmetric multiprocessing (SMP) nodes. Each node has four RS/6000 Power3-II processors (model 630), hence the total system contains 528 processors. The processors operate at a frequency of 375 MHz; two multiply-adds can be done per cycle resulting in a peak floating point rate of 1.5 GFLOPs per processor.

Not all of the 132 nodes can be used for computation. There are two nodes for interactive use; they are accessed automatically when commands are issued interactively.

Each of the 132 nodes have 4 GB of memory and 1 GB of local diskspace. In addition, the file system `/workspace` has 2.4 TB available and is mounted on all the IBM P3 nodes. While 520-fold parallel programs can in theory be executed on the ASC MSRC IBM P3, in practice, jobs requiring large numbers of processors will have to wait a long time before the required number of processors become available. It is thus more productive to use fewer processors to increase turn-around. See Section 4 for more information on submitting jobs to the queue.

2.2 IBM P3 File System Overview

2.2.1 IBM P3 `/workspace`

The `/workspace` filesystem is available to all IBM P3 users and contains 2.4 TB of disk space. This file system is mounted on all of the nodes through the IBM P3 high performance switch which provides bandwidth of over 130 Megabytes (MB) per second. The General Parallel Files System (GPFS) software provides software support for parallel file system operations. Currently there are six nodes that serve as parallel file servers. `/workspace` is intended for the temporary storage of data files needed for applications. Each user is given a directory named `/workspace/username` where *`username`* is the user's ASC MSRC login name. This directory will be created for the user at login via the ASC MSRC global `.cshrc` file whenever appropriate. An environment variable $WRK is set in the user's `.csrhc` file to set this up automatically.

There are no quotas on the amount of diskspace used in `/workspace`. `/workspace` is the only such area on the IBM P3, and applications ported from other systems should be modified to reflect this usage. $WRK is specified and local to the architecture and may differ from other systems. Currently files stored in a user's `/workspace` directory are deleted after a few days. Hence, users should consider data stored in this directory to be volatile. Users submitting jobs requiring input files stored on `/workspace` should not interactively copy them to `/workspace/username` and expect them to still be there when the job starts. Needed input files should be copied to `/workspace` as part of the batch job. The commands for

archiving the output can be found in Section 2.2.3. For the current ASC MSRC workspace policy see the Policies and Procedures web page at

http://www.asc.hpc.mil/.

The `/workspace` file system is **NOT** backed up or exported to any other system. In the event of file deletion or catastrophic media failure, such files are lost. **It is the user's responsibility to ensure that they save any data located in** `/workspace`.

2.2.2 /scratch2

The local filesystem, `/scratch2`, is defined on each of the nodes, and contains one gigabyte of storage. This file system may be used for storing executables or temporary storage for data. Files can be copied to `/scratch2` on each of the nodes using the `mcp` or `mcpscat` commands, while data can be retrieved using the `mcpgath` command. For more information on these commands see the man pages.

This file system is purged upon termination of the batch job, so users should be careful to copy important data to another file system before their job is completed.

2.2.3 Archiving Files

The ASC MSRC has an archival file server to allow users a convenient way to back up their files or to store files not currently needed. An environment variable $ARC is created in the `.cshrc` file as an alias for the user's home directory on the archive system. This server is transparent to the user, who can use the standard UNIX `cp` and `mv` commands to move files onto the archive file system, where they can be organized in a standard UNIX hierarchical file structure. The commands `cp` and `mv` are performed at normal network speeds, while `rcp` can use other interfaces, such as the ATM interface, to copy large files. In batch jobs the user is advised to use the non Kerberos version of `rcp` in `/bin/rcp` which does not require a Kerberos ticket. Users should consult the *ASC MSRC Archival Server User's Guide* for the directions on how to access the Archival and Storage system most efficiently.

2.3 IBM P3 AIX Operating System

The IBM P3 runs AIX 5.1 which is a superset of the UNIX operating system that is fully compliant with POSIX and XPG4. While AIX 5.1 has a 32-bit kernel, it has several 64-bit extensions for support of large file systems and large memory (>2 GB) programs. For more information on AIX, see the IBM web pages at:

http://www.rs6000.ibm.com/

2.4 Available Software

Software currently available on the IBM P3 includes FORTRAN, C, C++, MPI, and many third party software packages. A complete list of software is maintained on teh ASC MSRC web page (http://www.asc.hpc.mil/software/).

2.5 Development

The IBM compilers provide the tools needed to develop, compile, and analyze programs on the IBM P3 in FORTRAN, C, and C++.

The commands to invoke the FORTRAN compilers are `xlf` for sequential code and `mpxlf` for parallel programming using the MPI model. ASC MSRC also has `xlhpf`, IBM's high performance FORTRAN. The website http://www.asc.hpc.mil/IBM/ contains a set of manuals describing the MPI programming environment on the IBM P3. Users are strongly encouraged to become familiar with it.

`xlc` and `xlC` are the ANSI standard C and C++ compilers, respectively. These compilers can be invoked with a large variety of the commands users are familiar with from other systems. There are also man pages. Their MPI counterparts are `mpcc` and `mpCC`.

The following section describes the compilers and program development tools in more detail.

3. Program Development

Program devlopment in the IBM P3 computing environment is similar to that used in a typical UNIX environment. However, the user must take additional steps to utilize the multiple processors available.

3.1 Compiling and Linking

The compiler is an essential tool for software development. The features of the FORTRAN and C/C++ compilers are described below.

3.1.1 Serial Compilers

3.1.1.1 FORTRAN Compilers

The default FORTRAN compiler on the IBM P3 is the XL FORTRAN compiler. The serial FORTRAN compiler commands are `xlf, f77, xlf90,` and `xlf95`. These commands utilize the same compiler built with different default options (which are defined in the XL FORTRAN configuration file /`etc/xlf.cfg`).

The `xlf` and `f77` command are functionally identical and compile fixed-format FORTRAN 77 source code; however, since the base compiler supports FORTRAN 95, these commands can parse some FORTRAN 90 and 95 language constructs as well. The `xlf90` and `xlf95` commands compile free-format FORTRAN 90 or FORTRAN 95 source code. The default for all these commands is to compile standard FORTRAN with IBM-defined extensions. By compiling with the options `-qlanglvl=77std, -qlanglvl=90std,` or `-qlanglvl=95std` compiler options, the compilers can recognize conformance with the FORTRAN 77, FORTRAN 90, and FORTRAN 95 standards, respectively. Further information about the FORTRAN compilers can be found in the man pages or on the ASC MSRC IBM P3 documentation web page at http://www.asc.hpc.mil/IBM/.

3.1.1.2 C/C++ Compilers

The IBM Visual Age compiler is the default C/C++ compiler on the IBM P3. The commands `xlc,` or `cc` can be used to compile C programs while the commands `xlC` is used to compile C++ applications. Like the FORTRAN compiler the different C/C++ commands utilize the same compiler, but with different default options. The default options are described in the C compiler configuration file located in `/etc/vac.cfg`.

The commands `xlc,` and `xlC,` compile ANSI-standard source code, whereas `cc` compiles pre-ANSI source code with IBM-defined extensions. The `-qlanglvl` option can be used to change the language conformance of these commands.

Further information can be found in the man pages or on the ASC MSRC IBM P3 documentation web page at http://www.asc.hpc.mil/IBM/.

3.1.2 Parallel Compilers

3.1.2.1 FORTRAN Compilers

The standard FORTRAN compiler commands can be used to compile parallel Message Passing Interface (MPI) or Message Passing Layer (MPE) applications. However, IBM provides the `mpxlf` command to facilitate compiling and linking of parallel application. The `mpxlf` command ensures that the correct path to include files (such as `mpif.h`) can be resolved when compiling, and adds all necessary libraries when linking. The `mpxlf` is basically a FORTRAN 77 compiler like `xlf` and has the same defaults and supports all the same options. To compile MPI codes written in FORTRAN 90 free form compile and link with the `mpxlf`90 command**.**

High Performance FORTRAN (HPF) applications can be compiled with the IBM XLHPF compiler commands. The HPF commands are `xlhpf, xlhpf90,` and `xlhpf95.` The command `xlhpf` compiles fixed-format FORTRAN 77 source code, whereas `xlhpf`90 compiles free-format FORTRAN 90 source code, and `xlhpf`95 compiles free-format FORTRAN 95 source code. The default for all these compilers is to compile standard FORTRAN with IBM-defined extensions. With the `-qlanglvl` compiler option, the compilers can recognize conformance with the FORTRAN 77, FORTRAN 90, or FORTRAN 95 standards. Most of the compiler commands that are defined for `xlf` (1) will work with these compiler commands as well.

Further information can be found in the man pages or in the IBM Parallel Environment documentation located on the ASC MSRC IBM P3 documentation web page at http://www.asc.hpc.mil/IBM/.

3.1.2.2 C/C++ Compilers

The commands `mpcc` and `mpCC` should be used to compile distributed memory parallel C and C++ programs respectively. Like the `mpxlf` command, these commands ensure that the proper include paths can be resolved when compiling and appends the correct libraries when linking. Any of the compiler options accepted by `cc` and `xlC` will work with `mpcc` and `mpCC`, respectively.

Further information can be found in the man pages or in the IBM Parallel Environment documentation located on the ASC MSRC IBM P3 documentation web page at http://www.asc.hpc.mil/IBM/.

3.1.2.3 Parallel Compilers and the Parallel Operating Environment (POE)

One of the main differences between the executables built with the serial and parallel compiler commands is that the parallel commands (`mp*`) link in the MPI library and the Parallel Partition Manager when the linker/loader is invoked. When a code linked with the parallel compiler commands is executed, the program starts POE, and the program is run in parallel on all available processors even if the code is serial and does not contain any MPI routine calls. Hence, it is important to compile serial codes with the serial compiler commands and parallel message passing codes with the parallel compiler commands. The Parallel Partition Manager and Poe are discussed in more detail in Section 4.

3.1.3 Compiling SMP Programs

3.1.3.1 Reentrant Compiler Commands

The IBM P3 has SMP nodes which allow the user to exploit the shared-memory parallel programming model, or multi-threaded programming models within the node. To utilize the multi-threaded model the code must be compiled with the reentrant (thread-safe) versions of the compiler. The reentrant compiler commands have a `_r` suffix; for example, the reentrant version of the `xlf` command is `xlf_r.` The compiler commands include the compiler options to ensure that the code is thread safe and adds the `libpthread.a` library and thread safe versions of runtime libraries when linking.

It is important to note that the default storage class for local variables is static when using the reentrant form of the FORTRAN 77 compiler (`xlf_r`). It is recommended that the `-qnosave` flag be used with the `xlf_r.` command to change the default storage class to automatic. This will cause local variables to be allocated on the stack which is necessary to avoid storage conflicts in multi-threaded applications.

3.1.3.2 Auto Parallelization and OpenMP Directives

The `-qsmp=auto` compiler option can be used to instruct the compiler to automatically parallelize explicit **DO** loops (or **for** loops in C) and implicit **DO** loops created by the compiler when using FORTRAN array syntax.

Both the C and FORTRAN compiler support the OpenMP directive standard. Compiling with the flag `-qsmp=omp` instructs the compiler to recognize the FORTRAN `!$OMP` directive sentinel or the `#pragma omp` directive pragma in C.

Note: The `-qsmp=omp` will turn off auto parallelization; to use OpenMP directives and automatic parallelism, compile with the flag `-qsmp=omp:auto`.

3.1.3.3 Mixed MPI and Multi-threaded Programs

The multi-threaded and distributed memory programming models can be mixed on the IBM P3. For example the multi-threaded model can be used on a node and MPI can be used to send messages between nodes. The FORTRAN, C, or C++ compiler commands `mpxlf_r`, `mpcc_r`, or `mpCC_r`, respectively; must be used to compile distributed memory multi-threaded code. These commands will add the appropriate compiler flags and add `libpthreads.a` and the thread-safe libraries when linking.

3.1.4 Useful Compiler Options

3.1.4.1 Options Used for Debugging

By default, the IBM compilers do not trap signals for exceptions like floating point exceptions or array out-of-bounds errors. Checking for these errors is often useful when developing code or debugging. However, these options can slow execution of the code. Below is a list of some common options that can be used for debugging to identify where certain types of errors are generated in the code.

Table 1: Common Flags Used for Debugging

-qfltrap=options	Detect floating point exceptions at runtime. To enable checking for overflow, underflow, zero divides, and invalid floating point operations use **-qfltrap=ov:und:zero:inv:en**.
-qcheck, -C	Detects array out-of-bounds errors.
-qextchk	Detects type mismatches between common blocks, modules, and procedure interfaces at compile and link time.
-qinitauto=hex_value	Initializes each byte of storage for automatic variables to a specific value. To initialize automatic variables to negative Not a Number (-NaNQ), use **-qinitauto=FF**.
-qsigtrap	Without arguments, this option will ensure that floating-point exceptions and error signals are printed to standard error.
-g	Produces debug information that can be used by a symbolic debugger and provides for more detailed information about run-time error signals

Note all of the options described above are valid for the FORTRAN and C/C++ compilers (with the exception of `-qsigtrap` for the C/C++ compiler), although the semantics of the options for the C compiler may be slightly different for the two compilers.

3.1.4.2 Options Used for Performance Optimization

By default, the IBM compilers do not perform any code optimization. There are several compiler options that can be used to tune the performance of the application. Below is a list of options often used for performance tuning.

Table 2: Common Flags Used for Optimization

-O2	Performs basic optimizations that do not affect semantics of the code.
-O3	Performs aggressive optimizations that have the potential to affect the sequential semantics of the code.
-qhot	Performs loop optimizations. Note this flag must be used in conjunction with **-O2** or **-O3**.
-qarch=arch	Instructs the compiler to utilize instructions for a specific architecture denoted by the arch argument. Note, for the P3, chip the *arch* argument is pwr3.
-qtune=arch	Performs architecture specific optimizations. If used without the **-qarch** option this switch allows one to tune for a specific architecture, but not use chip-specific instructions.
-qipa	Performs interprocedural analysis.
-qinline, -Q	Instructs compiler to inline procedures.

In general, the most robust set of performance tuning options for the IBM P3 that will work for most codes are `-O3 -qhot -qarch=pwr3.` This set of flags uses the highest level of optimization and targets the P3 architecture. If this set of options produces errors when compiling or during execution consider removing the `-qhot` option and using `-O2`.

Users should be careful about using the `-qipa` and `-qinline` options. These options can degrade the performance on some codes and can affect the numerical results produced by the program.

Note: All of the options described above are valid for the FORTRAN and C/C++ compilers, although the semantics of the options for the C

compiler may be slightly different for the two compilers.

3.1.5 Creating an Executable Program

3.1.5.1 Compiling a FORTRAN Program

If the entire program is contained in a single file, the code can be compiled and linked in one line. For example, to compile and link a serial FORTRAN program `foo.f` to create an executable application `bar` that uses ESSL routines, type

XLF -o bar foo.f -lessl

In the above example *XLF* can be any one of the serial or parallel FORTRAN compiler commands.

3.1.5.2 Compiling a C/C++ program

To compile and link a C program `foo.c` to create an executable application `bar` that uses Engineering and Scientific Sub Routine Library (ESSL) and math library routines, type

XLC -o bar foo.c -lessl -lm

on the command-line; where *XLC* may be `cc`, `xlc`, or the parallel C compiler command mpcc.

The IBM C++ compiler recognizes files that end in .C as C++ codes. To compile and link a C++ program `foo.C` to create an executable application `bar` that uses ESSL and math library routines, type

XLC -o bar foo.C -lessl -lm

on the command-line, where *XLC* can be `xlC`**,** or `mpCC`.

3.1.5.3 Compiling Programs that Require More than 256 MB of Memory

The program data and stack segments are defined at link-time; the default limit is slightly less than 256 MB total for the data and stack segments. For programs that define large static arrays or allocate large dynamic arrays (C, C++, and FORTRAN 90), use the `-bmaxdata:bytes` linker option to increase the data segment of the executable. The bytes argument to the `-bmaxdata` option is the maximum amount of memory to reserve for the data segment and can be up to 2 GB (2 x 2^30 bytes).

For programs that use large amounts of automatic data or exceed the stack space limits, use the `-bmaxstack:bytes` linker option to increase the stack soft limit up to 256 MB.

Note: The `-bmaxdata` and `-bmaxstack` flags are linker options that are accecpted by the compilers. See the *ld* (1) man pages and the

chapter “Large Program Support”

(http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixprggd/genprogc/lrg_prg_support.htm) in the IBM documentation.

The `size` (1) command returns the size (in bytes) of the various memory segments in an executable. This command can be used to examine the size of the various data segments in the program and determine the maximum amount of memory a program will use (per processor) if the program does not perform any dynamic memory allocation. For example, consider a FORTRAN program `bigmem.f` which is compiled to produce the executable called `bigmem`. The `size` command produces the result

```
% size bigmem
bigmem: 1108+228+536870924+595+12 = 536872867
```

The `size` command displays the sizes of individual data segments in the binary executable (the description of these segments is unimportant for this discussion), and the sum of all data segments. In this case, the sum of all the data segments is about 540 MB; hence, the `-bmaxdata` option should be used to reserve over 540 MB of memory. For example, the following command reserves 10^9 bytes which is more than enough memory for the program to run.

```
xlf -bmaxdata:1000000000 bigmem.f -o bigmem
```

3.1.5.4 Calling FORTRAN routines from C and C++

Because the calling interfaces differ between FORTRAN (call-by-reference) and C and C++ (call-by-value), C programmers wishing to call BLAS and/or LAPACK routines should reference “Using IBM C and C++ Compilers with Other Programming Languages” in the C/C++ documentation on the ASC MSRC IBM SP documentation web page at http://www.asc.hpc.mil/IBM/.

3.1.6 Libraries

Several numerical and scientific libraries are available on the IBM P3. Some of the more important libraries are described below.

3.1.6.1 Math Library

The Math Library (`libm.a`) contains definitions of the mathematical functions defined in the FORTRAN, C, and C++ languages. Among these are the trigonometric functions.

When linking object files with `xlc, xlC, cc,` and `c89`, the math library is not linked by default; using `xlf, xlf90,` and `f77,` the math library is linked by default. Add `-lm` to the linker/loader command when compiling with the C and C++

compiler commands if using the functions defined in the standard header file `<math.h>`.

3.1.6.2 Basic Linear Algebra Subprogram (BLAS) library

The Basic Linear Algebra Subprogram (BLAS) library is a collection of routines for performing vector and matrix operations. BLAS 1 routines perform vector-vector operations; BLAS 2, vector-matrix operations; and BLAS 3, matrix-matrix operations. Further information about the BLAS library can be found at http://www.netlib.org/blas/.

Add `-lblas` to the linker/loader command to include BLAS routines in the executable application. BLAS routines are also included in the ESSL and LAPACK libraries described below.

3.1.6.3 Engineering and Scientific Subroutine Library (ESSL)

The Engineering and Scientific Subroutine Library (ESSL) is a collection of routines for performing linear algebra, matrix operations, solving linear algebraic equations, eigensystem analysis, Fourier transforms and related computations, sorting and searching, interpolation, numerical quadrature, and random number generation. The run-time libraries have been tuned for the RS/6000 architecture and for the specific uniprocessors as well. The library contains subsets of the BLAS and LAPACK libraries also described in this section. Further information can be found in the documentation on the ASC MSRC IBM P3 documentation web page at http://www.asc.hpc.mil/IBM/.

Add `-lessl` to the linker/loader command to include ESSL routines in the executable application. When calling ESSL routines from C or C++, include the header file `essl.h` to prototype the routines.

3.1.6.4 Linear Algebra Package (LAPACK) library

The Linear Algebra Package (LAPACK) library contains routines for solving matrix problems involving systems of simultaneous linear equations, least-squares fits of linear systems of equations, eigenproblems, and singular value problems. In addition, routines for matrix factorizations such as LU and QR factorization and singular-value decomposition (SVD) are also included. Further information on the LAPACK library can be found at http://www.netlib.org/lapack/.

Add
`-L/app/lapack/platforms/ibm3/lib -llapack`
(FORTRAN 77 library)

to the linker/loader command to use the LAPACK library.

3.1.6.5 The BLACS library

The BLACS library is designed to provide a portable collection of routines to distribute matrices on distributed memory platforms for use in parallel linear algebra routines. More information can be found at http://www.netlib.org/blacs/index.html.

Add

```
-L/app/scalapac/platforms/ibm3/lib -lblacs
```
(core BLACS library for the IBM P3)

```
-L/app/scalapac/platforms/ibm3/lib
-lblacsF77init
```
(FORTRAN 77 BLACS interface for the IBM P3)

```
-L/app/scalapac/platforms/ibm3/lib
-lblacsCinit
(C BLACS interface for the IBM P3)
```

to the linker/loader command to include BLACS routines in the executable application.

3.1.6.6 Parallel Engineering and Scientific Subroutine Library (PESSL)

The Parallel Engineering and Scientific Subroutine Library (PESSL) provides parallel versions of a subset of the ESSL subroutines discussed above. The subroutines perform matrix operations, eigensystem analysis, singular value analysis, Fourier transforms, and uniform random number generation. The run-time libraries have been tuned for use with the MPI library. The library contains subsets of the BLAS and ScaLAPACK libraries also described in this section.

Further information can be found in the documentation on the ASC MSRC IBM P3 documentation web page at http://www.asc.hpc.mil/IBM/.

Add `-lpessl` to the linker/loader command to include PESSL routines in the executable application.

3.1.6.7 Parallel Basic Linear Algebra Subprograms (PBLAS) library

The Parallel Basic Linear Algebra Subprograms (PBLAS) library is a parallel version of the BLAS that runs on distributed memory platforms. The names and calling interfaces of the PBLAS routines are similar to their BLAS equivalents. The library uses the BLACS and BLAS libraries. For more information, see the references at http://www.netlib.org/scalapack/index.html.

Add

```
-L/app/scalapack/platforms/ibm3/lib -lpblas
```

to linker/loader command to include PBLAS routines in the executable application.

3.1.6.8 Scalable Linear Algebra Package (ScaLAPACK) library

The Scalable Linear Algebra Package (ScaLAPACK) library is a parallel version of a subset of the LAPACK library for distributed memory platforms. The library contains routines for solving matrix problems involving systems of simultaneous linear equations, least-squares fits of linear systems of equations, eigenproblems, and singular value problems. In addition, routines for matrix factorizations such as LU and QR factorization and singular-value decomposition (SVD) are also included. ScaLAPACK uses the PBLAS and BLACS libraries.

Further information on the ScaLAPACK library can be found at http://www.netlib.org/scalapack/scalapack_home.html.

An online version of the ScaLAPACK User Guide is http://www.netlib.org/scalapack/slug/scalapack_slug.html.

Add

```
-L/app/scalapack/platforms/ibm3/lib -lscalapack
```

to the linker/loader command to include ScaLAPACK routines in the executable application.

3.2 Debugging and Profiling

In this section several tools for debugging and performance tuning are described. A more detailed discussion of the utilities is available in man pages and other on-line documentation.

3.2.1 dbx

The **dbx** utility is a terminal based symbolic debugger that can be used to debug serial programs and examine core files. This debugger can be used to set break points at selected lines of code or to run the code line by line, in addition symbolic variables can be displayed. In order to take advantage of the symbolic debugging functionality, the **-g** flag must be used to compile the source code and when linking to create the executable.

3.2.2 xldb

xldb is a debugger with an X-based graphical user interface. Operations similar to those in **dbx** described above are also available in **xldb**. In addition, **xldb** includes features for debugging multi-threaded applications. A list of valid command-line options is given when **xldb** is run without options. An on-line help facility for **xldb** is provided within the program.

3.2.3 pdbx

The **pdbx** debugger is a terminal based parallel debugger based on **dbx** with support for distributed memory parallel programs. As such, most of the commands available in **dbx** are available in **pdbx** with the additional option of specifying the tasks which execute the commands.

The **pdbx** debugger is invoked like the **poe** command (discussed in the Section 4) and supports all of the **poe** option.

pdbx program [program options] [poe options]

3.2.4 pedb

pedb is another parallel debugger similar to **pdbx**, but has a graphical user interface (GUI). **pedb** is a **poe** utility and supports all of the **poe** options.

3.2.5 TotalView

TotalView is a debugging utility with a GUI that supports debugging serial, multi-threaded, and distributed memory parallel programs. TotalView is a third party Commercial-Off-the-Shelf (COTS) software package unlike all the other IBM utilities discussed so far.

To run Totalview on a serial or multi-threaded program, simply type

totalview program

For a distributed memory parallel program type

totalview poe -a program [program options] [poe options] [totalview options]

3.2.6 prof

The **prof** command displays, for each external text symbol, the percentage of execution time spent in that function, the number of times that function was called, and the average number of milliseconds per call. In order to take advantage of the **prof** utility, the **-p** option must be used to compile the source code and link the executable. Additional information on the number of calls to a subroutine can be obtained if the **-g** flag is used as well.

When a program that has been compiled and linked with **-p** runs, a binary file `mon.out` is produced which contains the profile data. In order to generate a **prof** report for a serial program, issue the command

prof executable

For distributed memory parallel programs each processor writes a profile data file of the form `mon.out.N` where N is the node task id which ranges from 0 to one less than the maximum number of tasks for the job. To generate a prof report for a single task (say task n), issue the command

prof executable -m mon.out.n

Multiple profile data files can be supplied to the **prof** command; the resulting report will present a sum of execution time and function call counts for each function. For example, if there are four mon.out files, the user can issue the command

prof executable -m mon.out.0 mon.out.1 mon.out.2 mon.out.3

3.2.7 gprof

The **gprof** utility is similar to **prof**, but **gprof** also records the control flow history for a program which can be used to show the parents and children for each subroutine. In order to take advantage of the **gprof** utility, the **-pg** option must be used to compile the source code and link the executable. When the program has finished running, a file `gmon.out` is created which contains the profiling information. If the program is run for a distributed memory parallel program, then multiple files are created each appended with a processor task id.

To view a **gprof** report for a serial program, type

gprof executable

If multiple `gmon.out` files are listed at the end of the command, **gprof** will sum all of the execution time and function call counts for each function. For example, if a parallel program runs on four processors generates four gmon.out files, the command to view the sum of all the **gprof** data would be

gprof executable gmon.out.0 gmon.out.1 gmon.out.2 gmon.out.3

3.2.8 xprofiler

The **xprofiler** utility is a GUI for viewing **gprof** data. This utility allows the user to view and filter the call graph of the program allowing one to view the call graph for a subroutine or to filter out functions that use a small amount of time. The **xprofiler** command is used in the same manner as the **gprof** command.

3.2.9 Visualization Tool (VT)

The visualization tool (VT) is a **poe** utility that can be used to understand the performance characteristics of a parallel program. **vt** includes a trace utility that can be used to obtain information on the CPU and disk utilization of processor nodes, and a detailed information on the nature of the communication between processors. This information is presented in an graphical format that is easily interpretable.

To use **vt** the program must be compiled and linked with the **-g** flag. Also, the program must be run with the **-tracelevel 9** command line option, or by setting the **MP_TRACELEVEL** environment variable to 9.

When the program completes a file named `progname.trc` is created, where `progname` is the name of the executable. The VT tool viewer can be used to view the information in the tracefile by typing

vt -tracefile progname.trc

More information on VT can be found in the **vt** man page or on the web page for POE Operation and Use, Volume 2 located in the Parallel Environment section on the ASC MSRC IBM P3 documentation web page at http://www.asc.hpc.mil/IBM/.

4. Running Jobs on the IBM P3

4.1 Using the Parallel Operating Environment (POE)

This IBM POE consists of several commands that allow users to execute and debug programs on multiple nodes of the IBM P3. The `poe` command is used to execute programs on remote nodes and is invoked by

poe program [poe options]

where the program can be a parallel or serial program.

Note: The program must exist in the same path on all nodes.

The behavior of the parallel command can be influenced using environment variables or command line options. The `poe` options and environment variables are described in the man page for `poe`.

4.1.1 Interactive Use

Interactive use of the IBM P3 is for program development, including debugging and performance improvement, job preparation, job submission, and the processing and post-processing of data. Interactive processes are limited to a maximum of 15 CPU-minutes/processor and up to two nodes (eight processors). There is up to 4 GB of memory available per node, for a total of 8 GB per interactive job. These jobs run on the two nodes set aside for interactive use; jobs with larger requirements must be run in a batch queue.

In order to run on the interactive nodes, the user needs a host file that contains a list of the interactive nodes, and must specify the number of processors to be used. For example, if a user creates a host file named `hostfile` and wishes to run the program `foo` on two processors, the user may type the command:

poe foo -hfile hostfile -procs 2

A typical hostfile for the IBM P3 would contain the following lines:

```
hpc04-1
hpc04-1
hpc04-2
hpc04-2
```

4.1.2 Batch Use

The IBM P3 uses IBM's LoadLeveler scheduler for batch jobs submission and scheduling. The basic features of a LoadLeveler command file and the various LoadLeveler commands are described in the following sections.

4.1.2.1 Creating a Script File

Before using LoadLeveler you must create or modify a script file containing LoadLeveler directives and shell commands. Below is an example LoadLeveler script.

LoadLeveler example script

```
#!/bin/csh
#
# file:   your_code.cmd
#
# purpose: run the executable, your_code, under LoadLeveler
#
# remarks: This command file is (also) a csh file, which is not necessary
#          for LoadLeveler, but in this way, one can mix csh and LoadLeveler
#          directives.
#
#          A LoadLeveler directive begins with # @
#          directives following comments with MUST, ... must be given
#
#          Make sure not to redefine/set environment variables that are
#          already set with any of the LoadLeveler commands you end up using;
#          the equivalent statement holds for command line parameters to poe;
#          (if redefinitions are done consistently, it's simply redundant,
#           but if they are inconsistent then this is asking for trouble;)
#           loadl will propagate all set variables to poe.
#
#           Most likely, you will not have any need for setenv commands to csh
#           because the parameter COPY_ALL to '# @ environment" will copy
#           a set of default values for the MP_* environment variables into your
#           poe runtime environment, intended to give you the best throughput.
#
#
# Specify class. See list of available classes
# @ class = Small
#
# Specify job_type, one of (serial, parallel)
# @ job_type = parallel
#
#  MUST specify job time limit
# @ wall_clock_limit = hh:mm:ss
#
#  (for a parallel job)
# @ total_tasks = number-of-processors
#
# MUST leave this line in
# @ environment = ENVIRONMENT = BATCH
#
# Use a network statement to specify communication protocol
# @ network.MPI = css0, shared, US
# file name for my stdout
```

```
# @ output = your_code.out
#
# file name for my stderr
# @ error = your_code.err
#
# @ job_name = your_job_name
#
# @ restart = no
#
# Notify user about job status by e-mail
#
# notification is one of (always, complete, error, never, start)
# @ notification = always
# @ notify_user = your_ logname@where.you.get.mail
#
# Defines shell interpreter for script
# @ shell = /bin/csh
#
# MUST have '# @ queue' to finish job step
# @ queue

# cd to $WORK_DIR
cd $WORK_DIR

# copy big input file from archive to WORK_DIR
/bin/rcp ${msas}:$ARC/sub1/in.big $WORK_DIR

poe $HOME/your_code

# copy larger output file to archive
/bin/rcp out.big ${msas}:$ARC/sub1/out.big

# clean up files from $WRK
rm file.input in.big work.exe file.out out.big
```

4.1.2.2 Submitting LoadLeveler Jobs

The `llsubmit` command is used to submit a job to the LoadLeveler queue. To submit a job script called `yourcode.cmd`, type

llsubmit yourcode.cmd

LoadLeveler will return with a response similar to

llsubmit:The job "hpc04-1.asc.hpc.mil.4247" has been submitted.

where the form is <hostname>.<job ID> is the character string that identifies your LoadLeveler job.

4.1.2.3 Checking LoadLeveler Job Status

The `llq` command displays information about the jobs queued or running under PBS. Typing `llq` on the command line gives an output similar to

Id	Owner	Submitted	ST	PRI	Class	Running On
hpc04-1.4251.0	user-1	8/23 10:25	R	50	Large	d8n1s
hpc04-2.2182.0	user-2	8/19 14:20	R	50	Background	d3n13a
hpc04-1.4248.0	user-3	8/23 09:14	R	50	Medium	d1n11s
hpc04-2.2264.0	user-4	8/23 13:05	I	50	Small	
hpc04-2.2265.0	user-5	8/23 13:08	I	50	Small	
hpc04-1.4250.0	user-6	8/23 10:13	I	50	Small	

6 job steps in queue, 3 waiting, 0 pending, 3 running, 0 held.

Id is the truncated string returned by `llsubmit.` *Owner* is the username of the job owner. *ST* is the status of the job which can be *I*dle, *R*unning, *H*olding, or *C*ompleted. *Class* is the job class to which the job was submitted.

4.1.2.4 Deleting and Holding Jobs

A LoadLeveler job can be removed from the queue using the `llcancel` command. For example, typing

llcancel <hostname>.<Job Id>

removes the job identified as <hostname>.<Job Id> from the queue.

The `llhold` command can be used to place a job on hold and release a held job. Typing

llhold <hostname>.<Job Id>

places a the job identified by <hostname>.<Job Id> on hold. The `-r` option is used to remove a hold state from a job, e.g.,

llhold -r <hostname>.<Job Id>

releases the job identified as <hostname>.<Job Id> from hold status.

4.1.2.5 Multiple Userspace Jobs

The LoadLeveler *node* keyword defines the number of nodes allocated to a job. By default, LoadLeveler will start one executable task on each node. The current versions of the Parallel System Support Programs (PSSP) allows for multiple tasks to run on a node and use the fast user space communication protocol. Two

LoadLeveler keywords, *tasks_per_node*, and *total_tasks* are provided to allow one to allocate more than one task or process on a node.

The *tasks_per_node* key word is used to specify the number of tasks to distribute on each node. For example, the LoadLeveler stanzas

```
# @ node=2
# @ tasks_per_node=4
```

will allocate two nodes and four tasks per node for a total of eight tasks.

The *total_tasks* keyword defines the total number of tasks that will be started for a job. For example the LoadLeveler stanzas

```
# @ node=2
# @ total_tasks=8
```

will allocate four nodes to the job and start eight tasks or four tasks per node (assuming there are two processors per node). Note, in the above example eight tasks will always be started even if the number of tasks is 4, 5, 6, 7, or 8; only the number of tasks on each node will be different.

Note: At the ASC MSRC the `llsubmit` command has been modified so that if the job command file only includes the *total_tasks* keyword, without specifying the *node* keyword, the job will be assigned the minimum number of nodes necessary to run the job. For example, on a system with four processors per node, setting *total_tasks* to 16 will implicitly allocate four nodes to the job.

4.1.2.6 Available Classes

On the IBM P3 the wall clock limit for a job depends on the number of tasks a job requests. In order to enforce this policy several job classes have been defined. The job class can be specified by using the *class* keyword the LoadLeveler command file. By default the job is automatically assigned if no *class* keyword is specified. A list of the available classes and their limits is shown at http://www.asc.hpc.mil under Policies and Procedures->Batch Queuing Systems and Queue Definitions.

Note: The Background class can be used by users who do not have remaining allocated CPU hours in their account. However, jobs submitted with this class have a very low priority and will not be run unless there are no idle jobs in the queue.

5. Customer Service

5.1 Customer Service Center

For customer assistance, call the ASC MSRC Service Center at 1-888-MSRC-ASC (1-888-677-2272), (937) 255-0194, or DSN 785-0194 or send e-mail with a description of the problem to msrchelp@asc.hpc.mil. The support analysts will help with anything related to ASC MSRC: third party software, UNIX, the different ASC MSRC computers, etc. If you have any questions about the ASC MSRC, call the Service Center first. If your problem or question is beyond the scope of their expertise, they will refer you to the appropriate resource to resolve it.

5.2 ASC MSRC Support

In-depth technical inquiries and problems in the Service Center are forwarded to the ASC MSRC Customer Assistance and Technology Center (CATC), which pursues such inquiries and problems through resolution as rapidly as possible. The ASC MSRC CATC will attempt to determine the nature of the problem, then identify and coordinate whatever resources are needed to resolve the problem.

The ASC MSRC also conducts training classes, which provides an introduction to the ASC MSRC. Intermediate and advanced classes on selected topics are also periodically announced on the Programming Environment and Training (PET) section of the ASC MSRC homepage. Topics for such classes may be requested through the Customer Service Center.

The ASC MSRC CATC is ready to support in an advisory capacity any engineer or scientist who is (or potentially is) an ASC MSRC user.

5.3 ASC MSRC Website

The ASC MSRC website is the best source for current ASC MSRC information. To access the ASC MSRC website simply access this URL: http://www.asc.hpc.mil.

Some of the topics found on the website include:

APPLICATIONS
Short and long descriptions of current ASC MSRC applications
http://www.asc.hpc.mil/software/

SYSTEMS
Information on ASC MSRC servers and Archival Storage
http://www.asc.hpc.mil/hardware/

CUSTOMER SERVICE
Available Customer Services
http://www.asc.hpc.mil/customer/

ONLINE DOCUMENTATION
Listings of the ASC MSRC User Guides are available for viewing. Instructions are given on obtaining postscript versions.
http://www.asc.hpc.mil/customer/userdocs/

VISUALIZATION LAB INFORMATION
Current status and other information about the Visualization Lab.

http://www.asc.hpc.mil/sciviz/

TRAINING

Current course offerings and schedule

http://www.asc.hpc.mil/education/training/

FREQUENTLY ASKED QUESTIONS

Submit questions and read about various topics (such as “Customizing Your Environment”)

http://www.asc.hpc.mil/

POLICIES AND PROCEDURES

The latest policies regarding usage of the ASC MSRC resources.

http://www.asc.hpc.mil/overall/policy_procedure/

Appendix A. IBM P3 Usage Hints

The following are tips and hints for the effective use of the IBM P3.

A.1 Runtime Considerations

A.1.1 Batch use is strongly recommended

The interactive limit for processes is set to two nodes and eight processors and 15 minutes for each job. The system will automatically terminate programs that attempt to exceed these limits. The batch queues must be used for jobs that exceed these limits. Users with small production jobs will be warned if they abuse the interactive nodes.

A.1.2 Think about the number of processors you request

The greater the number of processors requested, than the longer the job will wait before that number of processors become available simultaneously. The user should therefore balance increased wait time against enhanced performance. The point of diminishing returns has set in if one is requesting close to 50% of the compute nodes.

A.2 Files and Filespace

A.2.1 File Management in `/workspace`

In order to prevent `/workspace` from becoming filled, a file scrubber is used to automatically remove old files from `/workspace`.

The `/workspace` filesystem is not backed up to tape. It is the user's responsibility to transfer files that need to be saved to a location that allows permanent storage. Two possibilities are the user's HOME directory space on the file server or the user's ARC directory on the archival storage system.

A.2.2 Archival Storage

ASC MSRC has an archival file server to allow users a convenient way to back up their files on tape or to store files not currently needed. An environment variable $ARC is created in the `.cshrc` file as an alias for the user's home directory on the tape archive system. This server is transparent to the user, who can use the standard UNIX `cp` and `mv` commands to move files onto the archive file system, where they can be organized in a standard UNIX hierarchical file structure. The commands cp and mv are performed at normal network speeds, while rcp can use other interfaces, such as the HiPPI channel or ATM interfaces, to copy large files. In batch jobs the user is advised to use the non Kerberos version of rcp in `/bin/rcp` which does not require a Kerberos ticket. Users should consult the *Archival Server User's Guide* for the directions on how to access the archival and storage system most efficiently.

A.2.3 Keep I/O local to the machine

`/workspace` is a file system local to the IBM P3 (i.e., not NFS-mounted

from a file server). I/O on NFS-mounted file systems is very slow. Users should copy their data files to their `/workspace` directory to avoid I/O on NFS-mounted file systems.

Running in /`workspace` is advantageous if large data files are created or read in the course of a program: there are no quotas, whereas `$HOME` is 400 MB. However, the user must back up his own files, as `/workspace` will be scrubbed periodically and no system backups of `/workspace` are made.

A.3 Helpful IBM P3-related Websites

The following site contains several useful links relating to software development and executing code on the IBM P3.

http://www.asc.hpc.mil/IBM/